Contents

[June 4, 2014 3](#_Toc470004640)

[ODF Read/Write Meeting with Yongcun Hu 3](#_Toc470004641)

[Files 3](#_Toc470004642)

[History 3](#_Toc470004643)

[File Format 3](#_Toc470004644)

[Sample Data 3](#_Toc470004645)

[Python 4](#_Toc470004646)

[Installing in Eclipse 4](#_Toc470004647)

[PyDev Setup 4](#_Toc470004648)

[Hello World! 4](#_Toc470004649)

[June 5, 2014 5](#_Toc470004650)

[PyUnit 5](#_Toc470004651)

[R for Eclipse 6](#_Toc470004652)

[Install for Eclipse 6](#_Toc470004653)

[Install R 6](#_Toc470004654)

[June 6, 2014 7](#_Toc470004655)

[R Tutorials and Testing 7](#_Toc470004656)

[Numpy, scipy & matplotlib 9](#_Toc470004657)

[June 16, 2014 9](#_Toc470004658)

[Progress on R 9](#_Toc470004659)

[Installing R Unit 9](#_Toc470004660)

[June 17, 2014 10](#_Toc470004661)

[RUnit Failure 10](#_Toc470004662)

[Testthat Success 10](#_Toc470004663)

[test\_dir() 10](#_Toc470004664)

[auto\_test() 11](#_Toc470004665)

[R CMD check ODF 11](#_Toc470004666)

[Writing Test Cases 11](#_Toc470004667)

[June 18, 2014 12](#_Toc470004668)

[R CMD check 12](#_Toc470004669)

[Unit tests with R CMD check path issue resolved 12](#_Toc470004670)

[TexLive 12](#_Toc470004671)

[June 19, 2014 12](#_Toc470004672)

[SVN 12](#_Toc470004673)

[June 26, 2014 13](#_Toc470004674)

[R Package Building 13](#_Toc470004675)

[sessionInfo() 13](#_Toc470004676)

[ls(“package:ODF”) 14](#_Toc470004677)

[July 8, 2014 14](#_Toc470004678)

[Success and Compliance 14](#_Toc470004679)

[August 21, 2014 15](#_Toc470004680)

[Python ODF 15](#_Toc470004681)

[Ordered Dictionaries 15](#_Toc470004682)

[Indexing Lists 15](#_Toc470004683)

[Reading and Writing 15](#_Toc470004684)

[Packaging 15](#_Toc470004685)

[Conclusion So Far 16](#_Toc470004686)

[R 16](#_Toc470004687)

[Python 16](#_Toc470004688)

[Recommendation 16](#_Toc470004689)

[December 20, 2016 – Python Distribution Revisited 16](#_Toc470004690)

# June 4, 2014

## ODF Read/Write Meeting with Yongcun Hu

Discussion for ODF Read Tool

### Files

There are two main files for the tool (read\_odf.m, define\_odf\_header.m) located in the ODS2 SVN repository

**read\_odf.m** - <http://vsnsbiosvn01/OSD/ODS2/Trunk/ODSTools/Read/read_odf.m>

Contains the code used to read a file into memory, then process it and store the results in a structure. The resulting structure object is then passed to other tools for additional processing. The file also contains functions required for producing error reports when data can’t be properly processed. Yongcun asked that if I think of, or find, additional errors that require handling that I report them to him so additional methods can be added to the Matlab version of the tool

**define\_odf\_header.m** <http://vsnsbiosvn01/OSD/ODS2/Trunk/ODSTools/Core/define_ODF_header.m>

The definition file specific constants for the read function. The definitions were split off into a separate file because uses would change the definitions in the read\_odf.m script when running into issues reading ODF files instead of adjusting the ODF files. This lead to a huge mess in the ODF read code and unstandardized ODF files that required special “forks” of the ODF read tool to process.

### History

The original ODF Read tool required as many as 28 separate tools to process an ODF file. Yongcun and Jeff Jackson spent a lot of time cleaning up the tools in the ODS2 toolbox to make them as self-contained as possible.

### File Format

Files start with a large header section that describes all the metadata for the file and data it contains. The data section is separated from the header section by the string “-- DATA --“. Data is tab delimited and uses single quotes to indicate that a space is part of the data and not delimitation.

### Sample Data

[\\dcnsbioNA01b\BIODataSvcArc\Archive\](file:///\\dcnsbioNA01b\BIODataSvcArc\Archive\)

Yongcun recommended starting with CTD data as it’s the most error prone. I tried to access sample data, but permission denied and asked Shelley Bond if I require her permission before submitting helpdesk request.

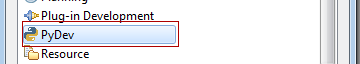
## Python

### Installing in Eclipse

Opened Eclipse and created a new workspace on the external drive “F:\Projects\MPR Assessment\python\”

Followed Tutorial for installing PyDev <http://www.vogella.com/tutorials/Python/article.html>

Install for Eclipse

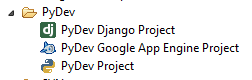
1. Help > Install New Software
2. Click Add
3. Name: PyDev
4. Location: <http://pydev.org/updates>
5. Click OK
6. Select “PyDev”
7. Click Next
8. Confirm “PyDev for Eclips”, Click Next
9. Accept licensing agreement
10. Install runs
11. “Do you Trust these Certificates? Brainwy Software: PyDev; Brainwy”
12. Select all
13. Click OK
14. Restart Eclipse
15. Open PyDev View
    1. 
    2. 
    3. 

### PyDev Setup

1. In preferences dialog, clicked New button next to “Python interpreters”
2. Interpreter Name: Pyton33
3. Interpreter Executable: C:\Python33\python.exe
4. Click Ok
5. Allowed all requested library paths to be added to the System pythonpath, just clicked OK
6. Clicked Ok to close preferences dialog, took a few minutes to configure

### Hello World!

1. Right clicked in Package Explorer
2. New Project
3. Selected PyDev Project



1. Next
2. Project name: “HelloWorld”
   1. Refer to <http://legacy.python.org/dev/peps/pep-0423/#use-a-single-name> for Python project name conventions
3. Had to setup the PyDev plugin to point to my Python installation at this point (See, “PyDev Setup” above)
   1. Project Type: Python
   2. Grammar Version: 2.7
   3. Interpreter: Default
   4. Add project directory to the PYTHONPATH
4. Clicked Finish
5. Right click “HelloWorld” project
6. Create new > source folder
7. Named it “src”
8. Click ok
9. Right click “src” folder selected “New PyDev Module”
10. Named: HelloWorld
11. Clicked Finish
12. Selected <Empty> in the Template selection
13. Clicked OK
14. Under the header comment added print “Hello Py World!!”
15. Ran as Python Run, Fail…
16. Google “Python Hello World” found out in python 3.0 print is a function and requires parenthesis. I’m using the Python 3.3 interpreter.
17. Changed print “Hello Py World!!” to print(“Hello Py World!!”)
18. Ran as Python Run, Success!!

# June 5, 2014

## PyUnit

Turns out PyUnit comes with PyDev by default, there was no additional requirements to set it up. I followed a Youtube tutorial (<https://www.youtube.com/watch?v=fU7RHewj6dg>), and the unit test worked as expected. I did have to fiddle around a bit when setting up the project to get a ‘src’ folder to be recognized as a folder and not part of the “package” path.

## R for Eclipse

Found this site (<http://navisan.com/articles/EclipseRHTML.aspx>) describing setting up StatET for R 3.3.0 in Eclipse.

## Install for Eclipse

Apparently Eclipse with Java has dependencies that conflict with Eclipse with R, so I had to install a blank version of Eclipse to install the R extension. I started a new work space in the same folder as the python workspace for R development.

1. From the Eclipse menu bar click Help -> Install New Software.
2. Click the 'Add' button. The "Add Site" dialog appears.
3. Type in a friendly name for your remote resource, such as StatET.
4. Paste the URL into the 'Location' box.
   1. I used (<http://download.walware.de/eclipse-4.3>) for Eclipse 4.3
   2. From <http://www.walware.de/goto/statet>
5. Click 'OK'.
6. Select (check) the package components that you want to install.
   1. I only chose StatET
7. Click 'Next'.
8. A review screen showing your selection(s) displays. Click 'Next'.
9. Accept the license agreement (if you agree to the terms described there).
10. Click 'Finish'.

## Install R

Downloaded R from r-project (<http://www.r-project.org/>) and installed version 3.1.0

1. Install R, if you need to. NOTE: as far as Eclipse is concerned, it makes no difference if you run 32 or 64-bit R. Just be sure that you identify 32 or 64-bit R to Eclipse, which we do in the next steps (among other things).
2. From the Eclipse menu bar click Help -> Cheat Sheets: StatET: R in Eclipse to put up the official help. This is a handy, built-in guideline, but some of the steps inaccurately describe the menu choices (and one reason why I wrote these instructions). It is a good reference, though, for your later work, so keep it in mind. Click the 'Cancel' button to close the Cheat Sheet Selection dialog.
3. From the Eclipse menu bar click Window -> Preferences: StatET : Run/Debug: R Interaction:
4. Select 'New Console Inside Eclipse' for the 'Connector used to run R code'. Note that this is the default so it might already be this way. This is how I run mine. Be sure to click 'Apply' if you change the setting.
   1. Back in the left pane click 'R Environments' (StatET : Run/Debug: R Environments, which is directly above the 'R Interaction' of the previous step). The 'R Environments' pane now appears on the right.
   2. Click the pane's 'Add' button. The 'Add R Environment Configuration' dialog appears.
   3. Type in a friendly name in the 'Name' box, such as R.2.11.1 or whatever your R version is, for example.
   4. For the 'Location (R\_HOME)' box, paste in the path to your R install directory, for example C:\Program Files\R\R-2.11.1.
   5. Make sure the 'Type' field is set to your R type (32 or 64-bit). D. Click 'OK' to save your changes and close the dialog, which returns us to the 'R Environments' pane, which now shows our newly created R environment.
   6. Click 'Apply' and 'OK' to save and close the 'R Environments' pane.
5. From the Eclipse menu bar click Run -> Run Configurations. The 'Run Configurations' dialog box appears.
   1. In the left pane click 'R Console' and define a new console entry with a friendly name in the 'Name' box, such as 'MyConsole'.
   2. Under the 'Main' tab ensure that 'Launch Type' is set properly for your configuration. If you are uncertain, choose 'Rterm' (this selection is VERY IMPORTANT).
   3. For 'Working Directory' just provide a path to your projects directory, for example C:\d2\SOFTWARE\_DEVELOPMENT\Research\R\Projects
   4. No further changes are required, but you can check to make sure that the 'R Environment' under the next tab ('R Config') is set to that environment you named in Step.4 above.
   5. Click 'Apply' and then 'Close' to close the dialog.

# June 6, 2014

## R Tutorials and Testing

R is working, I’ve been mucking through some of the tutorials found here <http://ww2.coastal.edu/kingw/statistics/R-tutorials/index.html> and it seems like a pretty straight forward language with a lot of potential. Graphs and Charts, much like in Matlab, seem really easy to create here’s some code and a sample:

# Creates a simple two line graph

#

# Author: upsonp

###############################################################################

#Create the plot area with 100x100 scale

plot(1:100, 1:100, type="n", xlab="", ylab="")

#draw a curve line on the plot

curve(x^2/100, add=TRUE)

#print the equation used to generate the a curve

text(80, 50, "This is a graph of")

text(80, 45, "the equation")

#notice how easy it is to nicely draw the equation

text(80, 37, expression(y == frac(1,100) \* x^2))

#denote some interesting points along the curve with triangle markers

points(c(20,60,90), c(4,36,81), pch=6)

#draw symbols down the right side of the graph

points(rep(100,10), seq(0,90,10), pch=0:9)

#draw a stright red line

abline(a=-18, b=1.1, col="red")

#draw some dotted lines on the graph

abline(h=20, lty=2)

abline(v=20, lty=3)

#Draw a square at the top center of the graph with

#circles as the corners

lines(c(40,40,60,60), c(80,100,100,80), type="b")

lines(c(40,60), c(80,80), type="l")

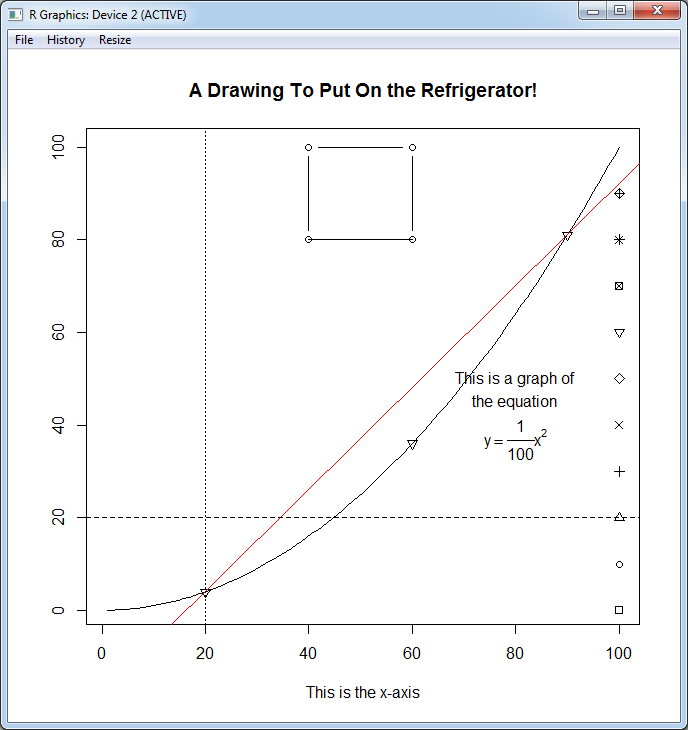
#title the graph

title(main="A Drawing To Put On the Refrigerator!")

#label the X-axis

title(xlab="x", col.lab="white")

title(xlab="This is the x-axis", col.lab="black")



## Numpy, scipy & matplotlib

Just had a discussion with Mitchel O’flaherty, he recommended several Python libraries that have some powerful chart/graph generations and data crunching abilities.

NumPy - <http://www.numpy.org/> is the fundamental package for scientific computing with Python. It contains among other things:

* a powerful N-dimensional array object
* sophisticated (broadcasting) functions
* tools for integrating C/C++ and Fortran code
* useful linear algebra, Fourier transform, and random number capabilities

SciPy - <http://www.scipy.org/> (pronounced “Sigh Pie”) is a Python-based ecosystem of open-source software for mathematics, science, and engineering. In particular, these are some of the core packages:

#### [NumPy](http://numpy.scipy.org) Base N-dimensional array package

#### [SciPy library](http://www.scipy.org/scipylib/index.html) Fundamental library for scientific computing

#### [Matplotlib](http://matplotlib.org/) Comprehensive 2D Plotting

#### [IPython](http://ipython.org) Enhanced Interactive Console

#### [Sympy](http://sympy.org/) Symbolic mathematics

#### [pandas](http://pandas.pydata.org/) Data structures & analysis

Matplotlib - <http://matplotlib.org/> is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. matplotlib can be used in python scripts, the python and [ipython](http://ipython.org) shell (ala MATLAB®[\*](http://matplotlib.org/" \l "ftn.matlab) or Mathematica®[†](http://matplotlib.org/" \l "ftn.mathematica)), web application servers, and six graphical user interface toolkits.

# June 16, 2014

## Progress on R

I made a lot of progress with R last week. I able to rewrite the define\_ODF\_header matlab file and started work on the actual read\_ODF matlab function. I used some homebrew Unit testing methods that seemed to work quite well and decided this morning I’d move into using an official unit testing package. In my case I decided to try out RUnit.

## Installing R Unit

I had some trouble figuring out how ti install R packages so that they can be referenced using the library(“packageName”) notation. After some googling around I discovered you can install official supported R packages by typing install.package(“packageName”) on the R command line, but when I tried it out I received a warning that my C:\Program Files\R\R-3.1.0 directory wasn’t writeable. Makes sense, on DFO machines C:\Program Files is a protected directory. I uninstalled R through the windows control panel > Programs and Features, then reinstalled R in C:\R\R-3.1.0. Restarted Eclipse and set the Runtime Configuration used to start the R Console to point to the C:\R\R-3.1.0 directory. Started the console and used the install.packages(“RUnit”) command, worked as expected. I also installed the BiocGenerics packages for additional function support

# June 17, 2014

## RUnit Failure

Oh pain and suffering, RUnit was a complete failure I couldn’t find enough documentation to get test up and running and what I did find was very obscure and recommended complex directory setups to get RUnit working.

## Testthat Success

After a lot of wasted time trying to get RUnit up and running I stumbled onto “testthat”, which is an alternative to RUnit. I installed it by typing “install.packages(“testthat”) in the R console in eclipse. This is my current setup:

R/ - for the application code

Read\_odf.R – containing the read\_odf() function and sub-functions

tests/ - for test files

test-all.R – file that includes the “testthat” library and notes it’s to test the ODF package

testthat/ - with the unit test files

test\_readODF.R – containing unit tests for read\_odf.R

testdata/ - to hold data files used in testing

… Various \*.ODF files used for testing different read\_odf() functionality

This has worked out exceptionally well. There are several ways to run the tests.

**REMINDER:**Before running either the test\_dir() or auto\_test() commands the testthat package must be imported into the R Console by typing library(testthat). Otherwise you receive an error message similar to “Error: could not find function "auto\_test"”.

### test\_dir()

In eclipse in the console use the getwd() command to make sure the current working directory is the root directory for the project. Use setwd(“<dir>”) to change the working directory if necessary. Then use test\_dir(“tests/testthat/”) to run all unit test in the testthat directory.

### auto\_test()

Another nice feature I’ve discovered that worked out really well was the use of “auto” testing. In eclipse, on the command line, use the command auto\_test(code\_path=”R/”, test\_path=”tests/testthat/”). Now whenever a file in either the R/ directory, where the application code resides, or the tests/testthat/ directory, where the unit tests reside, is saved auto\_test runs all the unit test automatically. FABULOUS!!

To stop auto\_test in order to re-engage the R console press “CTRL+BREAK” in Eclipse on windows, or “ESC” on a Mac.

### R CMD check ODF

The third way of running unit tests is using the “R CMD check ODF” on the command line, this is the MS DOS prompt command line, not the Eclipse R console command line. When distributing R Packages developers should run the “R CMD check <package name>” and “R CMD build <package name>” commands which will validate the R files in the package and produce a <package name>.tar.gz file that can be distributed to uses, which can then be loaded into R using the “install.packages(“<package name>.tar.gz”, repo=NULL, type=”source”)” command. Then functions from the package can be used by calling the “library(<package name>)” command. This would be the equivalent to downloading a Java API, adding it to the class path and using the “import org.dfo…..” to add classes and methods to the current class.

#### Test-all.R

To run unit test with the “R CMD check” command, the file tests/test-all.R needs to be created. The “R CMD check” command automatically checks this directory for files beginning with “test-“ and runs them. The test-all.R file looks something like this

------------------------------------------------------------------------------------------------------------------------------------------

library(ODF) # import ODF package with functions to be tested

library(testthat) # import testthat package

test\_check("ODF") # this is a testthat function that runs the tests located in the

# tests/testthat directory

------------------------------------------------------------------------------------------------------------------------------------------

## Writing Test Cases

Examples from ‘tests/testthat/test\_readODF.R’

Pretty easy to write test cases

Test\_that(“Some description of the test”, {

expect\_true(1 == 1)

expect\_false(1 == 2)

})

That’s pretty much it

# June 18, 2014

## R CMD check

I ran the “R CMD check ODF” command which failed on the unit test. The seems to be the “R/read\_odf.R” file wasn’t imported before the unit test was run. I had to add source(“F:/…/R\_ODF/R/read\_odf.R”) to the top of the /tests/testthat/test\_readODF.R file to get the test to pass, and it had to be the absolute path to the read\_odf.R file. That’s not a very optimal solution, there must be a way to include the R files. In any case after adding the source() line the unit test pass when running the “R CMD check ODF” command, but now it’s failing after, it has something to do with LaTex to PDF conversion.

### Unit tests with R CMD check path issue resolved

I was having an issue where the testthat tests failing when run with the “R CMD check ODF” command. Turns out all I had to do was add the line “library(ODF)” to my tests/test-all.R file. The “R CMD check” builds and installs the package, ODF in this case, temporarily, but it still needs to be imported into the work space using the “library(ODF)” or “require(ODF)” commands before functions from the package can be called.

### TexLive

Downloaded a zip and Installed TexLive from <http://www.tug.org/texlive/>

Wow, over an hour to install…

It seems to have worked, the “R CMD check ODF” command completed with one warning, no errors. The warning was because there’s documentation noting the read\_odf.R class is linked to write\_odf, update\_odf and validate\_odf, but I haven’t written those classes yet so… seems legit.

# June 19, 2014

## SVN

I’ve decided that I’m far enough along that I’d like to put the R\_ODF project into the svn at svn://vsnsbiosvn01/OSD/MPRAssesment/, visible on the web at (<http://vsnsbiosvn01/OSD/MPRAssesment/>). I installed Subversion in the Eclipse deployment I’m running R out of and had to also install SVNKit, it seems either Subversion or SVNKit might be causing an issue and Eclipse has been crashing all afternoon. I’ve wasted a ton of time just clicking something and waiting for a response only to have the crash report popup and the program to shut down. Looks like I’ll spend tomorrow re-installing Eclipse with R… Maybe I’ll just use TortoiseSVN to manage the R\_ODF project rather than risk spending a day clicking run and waiting 15 minutes to have the application crash.

On the plus side I’ve made significant progress with the read\_odf.R class. I’m looking over the matlab version and am wondering if all the additional code is really necessary. I’ll have to talk with Yongcun again about what the output is supposed to look like; because I can’t see any reason to require the define\_ODF\_header.R class unless it’s being used somewhere to validate the contents of an ODF file. The matlab code looks really inefficient in that there’s a double for loop with a lot of uncommented if statements that seem to be obscurely checking hbn (Header Name acquired from define\_ODF\_header.M) against various strings. I’m sure some logic could be applied to the if statements to cut out at least 20 lines of code from just the top of the matlab class, but I’m not a matlab programmer so maybe it’s a necessary evil of the language.

# June 26, 2014

## R Package Building

It’s been a long week, I’ve finally managed to get everything working just the way I think it should be. I still have some questions for Yongcun about the original matlab code, which I’ll outline below, but for the most part things are looking good. I imagine writing the write\_odf.R function should be a piece of cake now that I’ve pretty well got R down.

This week I focused mostly on getting read\_odf to produce a useful “ODF Object” that can be navigated and read by a user. In the process I’ve managed to figure out the intricacies of building a deployable package without all the issues of including the define\_ODF\_header.R class. It turns out there’s a @export tag in the documentation section of a class that’s used to make a function public. I was already doing that for the define\_ODF\_header function, but I didn’t use it for the read\_odf function.

The steps in building a package are as follows:

1. update the version number in the DESCRIPTION file at the root of the R\_ODF project
2. From the R console command line in Eclipse
   1. run **library(“devtools”)** to include the devtools package
   2. run **devtools::document()** to update the documentation and functions
3. from the MS DOS command line
   1. navigate to the directory above the R\_ODF project for me this was **F:\Projects\MPR Assessment\R>**
   2. Run the command “**R CMD check R\_ODF**” this will check the package and run the unit tests
   3. Run the command “**R CMD build R\_ODF**” this will build the ODF\_x.xx.tar.gz file, the version number is assigned based on the DESCRIPTION file modified in step 1
   4. Run the command “**R CMD INSTALL ODF\_x.xx.tar.gz**” replacing x.xx with the version number, this will install the ODF package
4. Back in eclipse at the R Console command line use **library(ODF)** to load the new ODF package
5. Typing **sessionInfo()**, should confirm the proper package has been loaded
6. Start reading ODF files with the command **odfObject <- read\_odf(“file/location/and/name.odf”)**

### sessionInfo()

I learned typing “sessionInfo()” on the eclipse R console command line will tell you what packages are currently loaded, it’s very useful. Here’s some sample output, this is when the R console is first started and no libraries have been included:

sessionInfo()

R version 3.1.0 (2014-04-10)

Platform: x86\_64-w64-mingw32/x64 (64-bit)

locale:

[1] LC\_COLLATE=English\_Canada.1252 LC\_CTYPE=English\_Canada.1252

[3] LC\_MONETARY=English\_Canada.1252 LC\_NUMERIC=C

[5] LC\_TIME=English\_Canada.1252

attached base packages:

[1] stats graphics grDevices utils datasets methods base

>

And here’s a sample after including the ODF library

> library(ODF)

> sessionInfo()

R version 3.1.0 (2014-04-10)

Platform: x86\_64-w64-mingw32/x64 (64-bit)

locale:

[1] LC\_COLLATE=English\_Canada.1252 LC\_CTYPE=English\_Canada.1252

[3] LC\_MONETARY=English\_Canada.1252 LC\_NUMERIC=C

[5] LC\_TIME=English\_Canada.1252

attached base packages:

[1] stats graphics grDevices utils datasets methods base

other attached packages:

[1] ODF\_1.09

>

Note that the ODF library is indicated under “other attached packages” and it gives the version number, which is useful for debugging.

### ls(“package:ODF”)

Another very useful command is “ls(“package:ODF”)”, this will print a list of available functions from the indicated package. For example:

> ls("package:ODF")

[1] "define\_ODF\_header" "read\_odf"

>

Great stuff, this is how I figured out read\_odf wasn’t being made a public function and that’s why I was having so much trouble getting it to work after installing the ODF package.

# July 8, 2014

## Success and Compliance

I had a lot of success last week finishing the write\_odf function. I’m up to version 1.11 and it appears both read and write odf are working correctly, for the most part. The output from the write ODF function isn’t \*exactly\* like the input files, most notably the PARAMETER\_HEADER > CODE doesn’t contain quotes in the input test files, but write ODF is printing the values out with quotes. I’m of the opinion that with quotes is the proper method as the code values are strings and string should be surrounded with quotes. There might be some issues concerning confusions around the “CODE” and “WMO\_CODE” fields. CODE is required for a PARAMETER\_HEADER, but sometimes “WMO\_CODE” is used in some legacy ODF files, so technically the legacy ODF files it’s used in, instead of the CODE field, are not compliant with the ODF format.

Well it’s time for a break; I’ll go back to the web project for a few days. When I come back from vacation, July 10-26, I have Python training with Learning tree (July 29 – Aug. 1). After which I’ll redo the read and write ODF functions using Python.

# August 21, 2014

## Python ODF

It’s been quite a break, I had the python training from July 29 – August 1 and have been pretty much working on the Python version of the ODF read/write functions since. It turned out to be significantly harder to do in python then it was in R. I guess that’s mainly my fault, I took the code as it was written in R and then changed the syntax and function calls to work in Python instead of just starting from scratch. Turns out that was just a major headache because of the differences in how R and Python handle arrays.

### Ordered Dictionaries

As an example you can’t assign “names” to Python arrays (lists) you have to use dictionaries in Python, but dictionaries are like hash tables and you can’t control the order in which things are stored in them. After a lot of headaches trying to write methods to store objects in-order in the dictionaries I stumbled on Ordered Dictionaries in the collections package, which helped significantly.

### Indexing Lists

The next issue I ran into was with indexing in R you can index a 2D-array using a notation like data[ :3 ] or data[ :’TEMP\_01’] indicating you want the value in the third column from all rows in the table OR all values in the ‘TEMP\_01’ column from every row. Something like the numerical method of indexing exists in Python, but I wasn’t able to get it to work properly. It might be part of the Scipy package I was told about several weeks ago, but I haven’t looked into it yet. As such I gave up on it and a significant section of code had to be re-written to work in python. Also not being able to name the columns in the Python version of the code makes it a little more difficult for the user to read the data.

### Reading and Writing

On the plus side Python was way more powerful when it came to parsing lines read from a file and it was a lot easier to write ODF objects out to a file.

## Packaging

I did manage to get python working with unit tests and was able to much more easily package the code for distribution. The python documentation for doing this was easy to find and easy to follow whereas most of the time I spent with R was spent researching how to package and distribute the code.

# Conclusion So Far

If I had it to do over I’d write the python code for read\_odf and write\_odf from scratch and use a more object orientated method since that would work in pythons favor. Starting from the R code was a bad idea because of the subtle differences and ended up being \*mostly\* a rewrite anyway. Both R and Python have some pretty strong features.

### R

Naming arrays and tables for R along with its much more script like language makes it easy to pick-up. Distribution and installing the packages are easy enough, but packaging was a pain to figure out and can be very touchy. R documentation is lacking and very hard to find, but has great built in support for a lot of scientific functionality if you can figure out how to use it.

### Python

Python has really great object orientation so you could feasibly develop much more complex programs. Structuring projects was a little tricky, but packaging them was a piece of cake. Python is also very well supported with great documentation and has lots of external modules for doing complex scientific work.

### Recommendation

If I had to make a recommendation for one language over the other, it would be for Python. Aside from being just as easy to pick up as R, my reasons are:

1. It’s a much more popular language than R, which has significant impact on the ability to get help from the community.
2. It has great documentation that makes it easier to research and discover methods/classes/modules that can be of use.
3. Finally, it has a lot of modules to do a lot of complex things including graphics, graphing and scientific calculations.

# December 20, 2016 – Python Distribution Revisited

It was brought to my attention that I never documented how distribution using pydev worked, It took me a bit to get setup, but it was ultimately pretty simple.

1. The parameters for the distribution, such as version number and project name, are laid out in the setup.py file
2. Open a command window and change directory to the python directory containing the project’s setup.py file. “E:\Projects\MPR Assessment\python\ODF” in my case.
3. Type: “python setup.py sdist”
4. A new \*.zip file should be created in the python/ODF/dist directory with the updated version number.